

Abstract

- The talk begins with the early days of Unix from my experience at Bell Labs and discusses some of the key decisions made designing the shell language.
- Why we did what we did. What worked and what changes were made early on.
- Some of the shell innovations will be discussed as well as what we learned later.

Early days of Unix and design of sh

Stephen R. Bourne

Rally Ventures and ACM Queue EiC

BSDcan, Ottawa

June 12, 2015

How I got to Bell Labs

- At Cambridge with Wilkes and Wheeler
- Algebra systems for Lunar Theory (CAMAL)
- Z language and life game (Conway '70)
- Algol68C compiler and ZCODE
- Cambridge CAP operating system
- Arrived at Murray Hill January 1975

Brunsviga at Mathematical Laboratory

Unix beginnings

- First Edition 1971
 - Running on the PDP 11
 - Hierarchical file system
 - Processes
 - Simple IO
 - Pipes (1972)
- BCPL, B, C
- Shell
 - Script recording with simple GOTO
 - `/etc/glob` for wild card characters
 - Shell scripts are not filters
 - No control flow or variables (`$1 - $9`, `$a - $z`)
- *troff* and *nroff*
 - Text processing funded Unix work
- The UNIX Time-sharing System - A Retrospective by Dennis Ritchie

Key Players

- Dennis Ritchie (C and drivers)
- Ken Thompson (kernel)
- Doug McIlroy (pipes)
- Joe Ossanna (nroff, troff)
- Brian Kernighan (C book)
- Mike Lesk (tbl, lex, uucp)
- Steve Johnson (portable C, yacc)
- Al Aho (awk, yacc)
- Stu Feldman (F77, make)

Unix Development

- Two adjoining rooms in 6th floor attic
- PDP 11/45
- Model 33 teletypes and Tektronix 4014
- uucp – no other network
- 300/1200/9600 baud modems at home
- Instant design and quality feedback
- Source code and man pages online
- Directory conventions (etc, bin, src, man, ...)
- RP05 disks and vertical surfaces

Sixth edition 1975

- Written in C
- 40 system calls
- grep man page was 20 lines
- Used outside of Unix group
- ed, grep, sed
- cc *.c, interfaces as .h files
- db, cdb
- Simple shell

TABLE OF CONTENTS

I. COMMANDS

ar	archive and library maintainer
as	assembler
bas	basic
bc	arbitrary precision interactive language
cat	concatenate and print
cc	C compiler
cdb	C debugger
chdir	change working directory
chmod	change mode
cmp	compare two files
comm	print lines common to two files
cp	copy
cref	make cross reference listing
date	print and set the date
db	debug
dc	desk calculator
dd	convert and copy a file
diff	differential file comparator
dsw	delete interactively
du	summarize disk usage
echo	echo arguments
ed	text editor
eqn	typeset mathematics
exit	terminate command file
fc	Fortran compiler
file	determine file type
find	find files
goto	command transfer
grep	search a file for a pattern
if	conditional command
kill	terminate a process
ld	link editor
ln	make a link
login	sign onto UNIX
ls	list contents of directory
mail	send mail to designated users
man	run off section of UNIX manual
msg	permit or deny messages
mkdir	make a directory
mv	move or rename a file
neqn	typeset mathematics on terminal
newgrp	log in to a new group
nice	run a command at low priority
nm	print name list
nohup	run a command immune to hangups
nroff	format text
od	octal dump
opr	off line print
passwd	change login password
pfe	print floating exception

pr	print file
prof	display profile data
ps	process status
pwd	working directory name
rc	Ratfor compiler
rev	reverse lines of a file
rm	remove (unlink) files
rmdir	remove directory
roff	format text
sh	shell (command interpreter)
shift	adjust Shell arguments
size	size of an object file
sleep	suspend execution for an interval
sort	sort or merge files
spell	find spelling errors
split	split a file into pieces
strip	remove symbols and relocation bits
stty	set typewriter options
tee	pipe fitting
time	time a command
tp	manipulate DECTape and magtape
tr	transliterate
troff	format text
tty	get typewriter name
typo	find possible typos
uniq	report repeated lines in a file
usort	sort and merge files, discarding duplicate lines
wait	await completion of process
wc	word count
who	who is on the system
write	write to another user
yacc	yet another compiler-compiler

II. SYSTEM CALLS

intro	introduction to system calls
break, brk, sbrk	change core allocation
chdir	change working directory
chmod	change mode of file
chown	change owner and group of a file
close	close a file
creat	create a new file
csw	read console switches
dup	duplicate an open file descriptor
exec, execl, execv	execute a file
exit	terminate process
fork	spawn new process
fstat	get status of open file
getgid	get group identifications
getpid	get process identification
getuid	get user identifications
gtty	get typewriter status
indir	indirect system call
kill	send signal to a process

link	link to a file
mknod	make a directory or a special file
mount	mount file system
nice	set program priority
open	open for reading or writing
pipe	create an interprocess channel
profil	execution time profile
ptrace	process trace
read	read from file
seek	move read/write pointer
setgid	set process group ID
setuid	set process user ID
signal	catch or ignore signals
sleep	stop execution for interval
stat	get file status
stime	set time
stty	set mode of typewriter
sync	update super-block
time	get date and time
times	get process times
umount	dismount file system
unlink	remove directory entry
wait	wait for process to terminate
write	write on a file

III. SUBROUTINES

abort	generate an IOT fault
abs, fabs	absolute value
alloc, free	core allocator
atan, atan2	arc tangent function
atof	convert ASCII to floating
atoi	convert ASCII to integer
crypt	password encoding
ctime, localtime, gmtime	convert date and time to ASCII
ecvt, fcvt	output conversion
end, etext, edata	last locations in program
exp	exponential function
floor, ceil	floor and ceiling functions
fmod	floating modulo function
fptrap	floating point interpreter
gamma	log gamma function
getarg, iargc	get command arguments from Fortran
getc, getw, fopen	buffered input
getchar	read character
getpw	get name from UID
hmul	high-order product
ierror	catch Fortran errors
ldiv, lrem	long division
locv	long output conversion
log	natural logarithm
monitor	prepare execution profile
nargs	argument count
nlist	get entries from name list

perror, syserrlist, sysnerr, errno	system error messages
pow	floating exponentiation
printf	formatted print
putc, putw, fcreat, fflush	buffered output
putchar, flush	write character
qsort	quicker sort
rand, srand	random number generator
reset, setexit	execute non-local goto
setfil	specify Fortran file name
sin, cos	trigonometric functions
sqrt	square root function
ttyn	return name of current typewriter

IV. SPECIAL FILES

cat	phototypesetter interface
dc	DC-11 communications interface
dh	DH-11 communications multiplexer
dn	DN-11 ACU interface
dp	DP-11 201 data-phone interface
hp	RH-11/RP04 moving-head disk
hs	RH11/RS03-RS04 fixed-head disk file
ht	RH-11/TU-16 magtape interface
kl	KL-11 or DL-11 asynchronous interface
lp	line printer
mem, kmem, null	core memory
pc	PC-11 paper tape reader/punch
rf	RF11/RS11 fixed-head disk file
rk	RK-11/RK03 (or RK05) disk
rp	RP-11/RP03 moving-head disk
tc	TC-11/TU56 DECtape
tm	TM-11/TU-10 magtape interface
tty	general typewriter interface

V. FILE FORMATS AND CONVENTIONS

a.out	assembler and link editor output
ar	archive (library) file format
ascii	map of ASCII character set
core	format of core image file
dir	format of directories
dump	incremental dump tape format
fs	format of file system volume
greek	graphics for extended TTY-37 type-box
group	group file
mtab	mounted file system table
passwd	password file
tabs	set tab stops
tp	DEC/mag tape formats
ttys	typewriter initialization data
utmp	user information
wtmp	user login history

Seventh edition and BSD

- 1976
 - Wrote sh and adb
- 1977
 - make, lint, awk, uucp
- 1978
 - 32 bit port and Berkeley
 - Seventh edition published
 - File system improvements
 - 32-bit port to Interdata
- 1977 – 1980
 - Bill Joy vi and the C shell
 - BSD virtual memory
 - BSD 3.0 and overnight install

Seventh edition release management

- Utility owned by last person who touched it
- Makefiles for everyone
- Some attempts to de-lint
- man pages, src, executables, libraries
- Directory structure
- But no source control or versioning
- Ran compiles from release tree

Why we started over and re-wrote sh

- Meeting in December in Murray Hill with Dennis
- Ken Thompson was in Berkeley for a year
- PWB Mashey shell started mid 1975
- All were patched versions of osh
- Started writing code toward end of 1975
- First versions deployed in early 1976

sh as a language

- Typeless
- Strings are first class and only citizens
- Delimited by blanks and reserved characters
- Serves as interactive and scripting language
- Provides programmable interface to the Unix system
 - Variables and substitution
 - Control flow
 - Signal management
 - Process management
 - eval

Shell Design and implementation

- ALGOL 68 concepts
 - Program flow
 - Closed forms (if ... fi, case... esac)
 - Complete substitutions anywhere
- No limits on strings (or anything else)
- String quoting rules
- Return values used for conditionals (exit=)
- Performance and strings
- Memory allocation

Shell features

- Multi character variables
- Environment variables
- Here documents (<<) with EOF and substitution
- Scripts as filters
- Command substitution
- Commands as functions (later)

Complete list from 1977

- dynamic storage management (for variables &c.)
- control structures (at tty as well)
- parameter substitution + - ? =
- case pattern matching
- path searching
- built in file name matching (/etc/glob)
- no re-evaluation after \$ substitution
- more general patterns (/sys/s?/...)
- general trap and fault handling
- control over child signals
- piping into or out of loops
- mail notification
- argument substitution eg nohup(a;b)
- efficient for loop and reads from files
- error recovery from sub command failure assured
- shell error handling can be controlled
- checks for cat >x >x and cat >x | wc
- cannot execute/cannot find distinguished
- general redirection e.g. 2>...
- input and execution traces; also no execution mode
- wait is interruptible
- cat >x hangs on open can be INTR'd

Unix at the time

- In 1975 we were running sixth edition Unix
- Did not use the C library
 - Mostly because I didn't need to
 - And also because of malloc conflicts
- Considered Yacc and lex but too heavyweight
- No stdio or string copy routines
- Directory layout fixed size
- No setjmp and longjmp (added)

Memory management

- Overall flow
 - Read in element
 - Evaluate from internal tree recursively
 - Same from tty or file
- During evaluation anything goes
 - Language is recursive so C stack not helpful
 - Variable assignment of arbitrary length strings anywhere e.g.

```
X = ` cat <<!  
    cd `pwd`  
    for v do read v; echo $v; done  
    !  
    ,
```

- Interleaved heap and stack
- Stack used for permanent objects including parse tree and partially constructed strings
- No string length or any other "arbitrary" length restrictions

Memory Management credits

- A Partial tour through the UNIX shell
 - Geoff Collyer, University of Toronto
 - Suspect this based on AT&T system 3 shell
 - Based on bugs reported
- Annotated source by Akira Nakamura 1990

Memory allocation

(upside down)



The diagram consists of two vertical lines. The top line has a downward-pointing arrow at its base, indicating memory allocation from the bottom of memory upwards. The bottom line has an upward-pointing arrow at its base, indicating memory allocation from the top of memory downwards.

sbrk from bottom of memory
sh heap storage at bottom of this area
stack for strings, command parsing at top
stack and heap are interleaved

C stack from top down

Memory allocation

- 
- A vertical line with a downward-pointing arrow on the left side and an upward-pointing arrow on the right side, indicating the direction of memory growth for the heap and stack respectively.
- ⌞ Base of heap at bottom of memory
Interleaved with stack items awaiting recovery
 - ⌞ stakbsy # chain of stack blocks that have become
covered up by heap allocation. `tdystak`
will return them to the heap
 - ⌞ Top of heap
 - ⌞ stakbas # base of entire stack
 - ⌞ stakbot # base of current stack item
 - ⌞ staktop # top of stack
 - ⌞ brkend # top of entire stack
 - ⌞ C stack from top of memory down

Stack and heap

```
/*
 *      UNIX shell
 *
 *      S. R. Bourne
 *      Bell Telephone Laboratories
 */

/* To use stack as temporary workspace across
 * possible storage allocation (e.g. name lookup)

 * a)      get ptr from 'relstak'
 * b)      can now use 'pushstak'
 * c)      then reset with 'setstak'
 * d)      'absstak' gives real address if needed
 */

#define      relstak()      (staktop-stakbot)
#define      absstak(x)    (stakbot+Rcheat(x))
#define      setstak(x)    (staktop=absstak(x))
#define      pushstak(c)   (*staktop++=(c))
#define      zerostak()    (*staktop=0)
```

The hard bits

- Signals and process management
cat <<! &
xyz
!
- Quoting specification and exclamation
- Error recovery and reporting
- Debugging memory allocation
- Here documents
- Performance
- The usual corner cases

Quoting hell

- There are at least three conceptually distinct mechanisms
 - \$ parameter substitution
 - argument splitting and parsing commands
 - file name generation (ls *)
- Suppose \$1, \$2 and \$3 have the values
 - \$1 = <>
 - \$2 = <a b>
 - \$3 = <*>
- Some choices are

arg	nsh	osh
\$1	-	-
\$2	<a>	<a>
\$3	<...>	<...>
"\$1"	<>	<\$1>
"\$2"	<a b>	<\$2>
"\$3"	<*>	<\$3>
"\ \$1"	<\$1>	<\$1>
'\$1'	<\$1>	<\$1>

Meta character rules

	\	\$	`	'	"	*	/
'	n	n	n	t	n	n	n
`	y	n	t	n	n	n	n
"	y	y	y	n	t	n	n

Unix group conversion

- printing exit= after each command
- od (octal dump) hence “done”
- goto gone
 - “The absence of a goto is going to be mourned loudly by many at BIS. We have a lot of COBOL and Assembler types here who don't seem able to live without it.” (1977 comment)
- wait command interruptable
 - “An interruptable wait has been long awaited and welcomed. Maybe it should have a flag argument to make it uninterruptable if desired.” (1977 comment)
- sh scripts as filters (what it used to be)
- 32 bit porting 1978 and *0
- sbrk and more porting grief (things you don't think of when you write the code)
 - asked Dennis one day what sbrk recovery was in fault routine
 - never well documented

Language battles - C vs sh

From srb Sat Mar 5 13:17:31 1977

More seriously wrt do ... done

Since there seems to be no hope of C becoming an expression language (in which case it would be potentially more suitable for the shell) there is equally no hope that the shell and C will be the same language.

The worst problem is the if...else problem which in C requires a lookahead. The shell cannot afford to do lookahead since it is an interactive language.

‘word ‘ – ‘drow’ dangling ‘else’ and keeping if ... fi

- Ways of dealing with the word drow “problem”

```
case a in ... esac  
for i do ... od  
for i in a b c do ... od  
while c do ... od
```

```
case a { ... }  
for i { ... }  
for i in a b c { ... }  
while c { ... }
```

- Separating for and while (originally one construction) is reasonable in the command language context.
- Its use in programming languages is to look for a member of a set with some specific property; I cannot think of a use for this in the shell.

‘word’ – ‘drow’ dangling ‘else’ and keeping if ... fi

For all of the above both forms can easily be provided.

```
if c
then d
else e
fi
```

```
if c
      d
else e
```

Is the dangling else is better than the fi. One way out of dangling ‘else’ is to say that there are two constructions one with no else part and the other with a ‘then’ part

```
if c
      d
```

```
if c
then d
else e
```

‘then’ signals the coming of the else (so to speak). The dangling else has gone away but one has to remember that ‘then’ is written if there is an else part. It does avoid the lookahead problem.

Keeping if ... then ... else

It seems rather radical to remove both the 'structured programming' primitives (if, while). They are both used at present. The alternative for 'if' using 'select' is somewhat cumbersome and unintuitive.

Using && and || produces even more amazing looking programs. e.g.

```
if a; then b; else c; fi
```

Becomes

```
{{{a&&{b;set $rc=$r; true}}||c; set rc=$r;}; test rc = 0}
```

Keeping case ... esac

The choice of words (select, switch, case, ...) seems unimportant although of these 'case' is shortest.

The proposed construction is more error prone and is likely to give surprising results. e.g.

```
case $1 { ... }
```

will distribute the case inside the {...} so that leaving out the ; is a disaster.

Also how does

```
case $1; {-x
          -y
          echo x or y
          }
```

parse? There is no natural way to extend to more than one case selector.

At present there is enough redundancy to find mistakes.

Associative memory aka Environment variables

- “The shell has had a keyword mechanism working from within the shell (the wrong place) for a couple of years.”
- “The reason for introducing environments was to make this mechanism uniform for shell procs and C programs.”
- Named variables and context
 - Debates over sh conventions
 - `x=y cmd ...`
 - `cc`, `make`, and `dd` used `x=y` in its arguments
- You don’t have to know variables at each process level
- Unix process rules - no child surprises
- Keyword parameters should be settable either at the call of a `cmd`, or in an enclosing environment.

Environment variables

- A pcs should be able to have 'local' names. i.e. names which are not passed on to children nor cause the behaviour of the child to be modified in any way.
- The names used by a child can be set by any ancestor of the child. It should not make any difference to the child how the name was set.
- It should be possible to set a name for use by descendants.
- A pcs should not have to be a postman for all the names which are being passed. It only need look at those intended for it. Others will automatically be passed on to children.

Environment variables – alternatives

Wed May 17 09:04:24 1978

Viewed from a C routine there are four levels of scope:

- a) Automatic variables, whose life is the subroutines';
- b) External variables, whose life is the memory load;
- c) Shell environment variables, whose life is the session;
- d) Files, which last forever.

It would be nice if all of these looked the same. For example:

- a,b, and d are binary; c is string
- only c has "export" and "readonly",
- a,b, and d are accessed by address;
- c has associative searches for "x=".

We could have a storage class "session" which was like "extern" except it lives longer.

Unfortunately, this requires enormous system changes.

So make session variables look just like files, i.e. that a new set of routines sopen, sseek, sclose, sread, and swrite exactly image open, seek, ... except that they work on session files.

Environment variables

Fri May 12 01:44:05 EDT 1978

The shell has been modified to take advantage of the new UNIX environment passing stuff (notably `execve`).

As a consequence shell variables are initialized from the environment but are not transmitted back to the environment by default.

`export name`

causes `name` to be sent to the environment when processes are created

Early development - quoting

Thu Jun 3 18:21:29 EDT 1976

- In response to popular demand a number of changes have been made to the shell. Please let me know if there are any problems (srb).
- \$ is no longer used as the quoting character; instead a \ is used. For example
 - echo \; will print ;
 - echo \\$ will print \$
 - Within double quotes the meaning of \ is somewhat modified and the only characters escaped are " \$ and newline.
- A new quoting mechanism '...' has been added that inhibits all interpretation of the enclosed string. For example
 - echo '\$1' will print \$1
 - echo '\\' will print \

Early development

- 'here' documents are processed in one of two ways depending on whether the string following << is quoted or not.
 - not quoted (e.g. <<!)
 - a \ is used to escape a newline, \$ and the first character of the terminating string.
 - Parameter substitution occurs within the document.
 - quoted (e.g. <<!\!) all characters are passed literally and no parameter substitution occurs.
- Command substitution is now written `command`
 - Nested uses now need to be quoted.

Early development

- Built-in names \$- \$\$ \$# \$! \$?
- "\$@" is the same as "\$1" "\$2" ...
- Error checking e.g.
The notation \${...} is now checked more thoroughly.
Also the default strings are only evaluated when needed so that e.g.
 echo \${d-`newfile`}
 only executes the command newfile if \$d is not set.
- Interrupt handling e.g.
 - echo */*/ is now interruptable.
- Keyword parameters only before cmd
- Performance improvements

Fri Mar 4 17:02:57 EST 1977

Some modifications have been made to the shell.

One result is that it now runs up to twice as fast as the current shell. It also uses less space. As usual some bugs have gone and some more have appeared (although in both cases they escape my test programs)

Selling PWB group

- Our situation is as follows:
 - We would like to end up with the same sh as you people use, as modified only slightly for local system peculiarities (like the use of logtty(), logdir(), and logname()) to get at the extra 32 bytes in the u area).
 - We are really hurting on systems B and D because of the number of processes generated. The shell I've been working on could be installed any time, would be completely upward compatible with what we've got, but much faster for shell procs. There is therefore a great desire to do this until the hardware is upgraded (in several months).
 - Given the extent of sh use, and the fact that many projects around here use it for their daily work in deadline-driven situations, we simply cannot live with a traumatic conversion, unless that conversion:
 - **Gains greatly in performance, compared with the upward-compatible sh that we could install**
- Or
- **Gains greatly in expressive power, again, compared with what we could install in an upward compatible way**

In line (here) documents – PY shell (Mashey) debates

The new shell provides a notation for in line.

At present the MH shell makes a temporary file copy of the information not made by the PY shell.

- a) an EOF indication is given to the subcommand; (which means it can do block reads and there is only one EOF convention)
- b) the shell can recover from an error in the execution of the subcommand
- c) parameter substitution is available without change to the notation
- d) a shell file can be parsed without the need to execute the subcommands

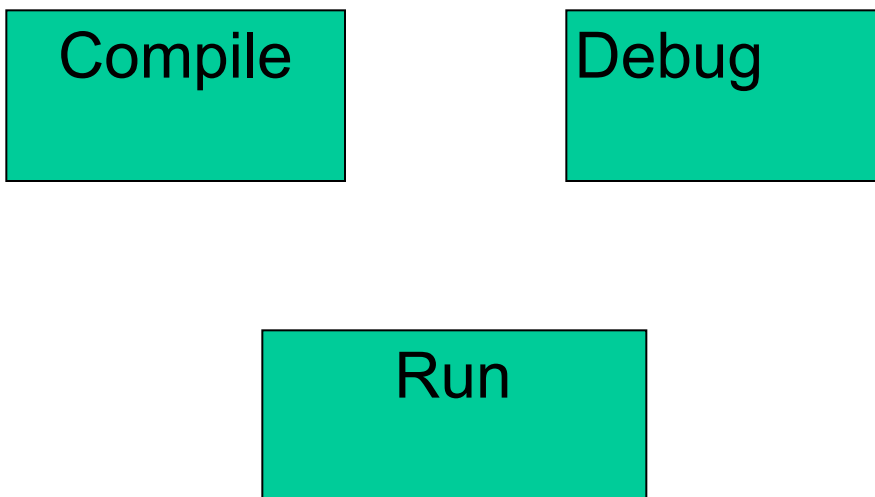
This copy can be avoided in the MH shell without changing the notation but in one case this will not provide an EOF indication to the subcommand and the implementation is complicated.

PWB shell differences

- a) After parameter substitution psh interprets syntactic characters, rsh does not. rsh provides `eval' for re interpretation.
- b) `\\" \"$` `\\" \\ \newline \"$ \``
- c) `$$` `$pid`
- d) `<-- pump` `<<!`
- e) control structure; mostly straightforward except for goto and switch without breaksw.
- f) `= a b c` `set ...`
- g) `.path`
- h) `exit default 0 default $r`
- i) `onintr` `trap`
- j) operator precedence of `&& ||` different
- k) `fd2` `2> 2>>`
- l) `glob` no match as is
- m) `$ax` `${a}x`
- n) `sh -t !( error` `!( ok`
- o) prompting rules are different
- p) `$t` set not set
- q) `chdir` no default default `$h`
- r) `next` .
- s) opt different flags setting rules (+x -x)

adb – a debugger

- Written to debug Algol68C port
- Added overlays via exec
- adb could run on one machine debug another and be compiled on a third
- How to separate out representations



The C language and libC

- Types
 - Structures like Algol68 not PL/1
 - Void
 - Pointers with explicit dereference
 - Casts
 - L and R values
 - Unions added
- Functions with local and global variables
 - No recursion
 - All functions returned a value
- Memory management
 - malloc, free
 - Strings done “by hand” no 'strcpy'
 - No array bound checking
- libC then stdio (later)
- Interfaces via .h files
 - Conventions for initializing global variables
 - Where instances are created
 - define vs declare

Usage by langpop

Various shells

- Original Thompson shell
 - 3 man pages
- sh
 - 6 man pages
- bash man pages and word count
 - 110 man pages
 - 4890 37094 329778
- POSIX and ksh
 - Early 80s
- dash
 - A return to sanity ?

Shell shock

- Suspect code written in 1993
- Unsuccessful in reaching authors
- Was it a simple error in using command line reader
 - Even that is a bug
- Most of environment (e.g. Apache) not present in '93
- Why did it take so long to surface – or did it

What worked and what I would have done differently

- Write code others can read (A68 macros)
- Memory management fragile when porting
- Functions as first class elements
- Add some real support to debug scripts
- Using 8th bit of byte as quoted character marker
- Start somewhere and iterate
- Have real users and hear what they say
- Resist feature creep
 - The shell can only solve so many problems
 - `cat -v` considered harmful (Pike)

Now what

- ACM Queue
 - Publishing for 10 years now
 - Focus is on problem not solution
 - Moving upstack
 - Will publish every other month via app
- Algol68C
 - Finish port
 - Incorporate libraries
 - Open source release

Thank you and Questions

Functional interfaces

```
STKPTR getstak(usize)
    INT      usize;
{    /* allocate requested stack */
    REG STKPTR  oldstak;
    REG INT      size;

    size=round(usize,BYTESPERWORD);
    oldstak=stakbot;
    staktop = stakbot += size;
    return(oldstak);
}
```

```
STKPTR locstak()
{    /* set up stack for local use
    * should be followed by 'endstak'
    */
    IF brkend-stakbot<BRKINCR
    THEN  setbrk(brkincre);
        IF brkincre < BRKMAX
        THEN  brkincre += 256;
        FI
    FI
    return(stakbot);
}
```

```
STKPTR cpystak(x) STKPTR x; {
    /* Copy a string onto the stack and allocate the space */
    return(endstak(movstr(x,locstak())));
}
```

Functional interfaces

```
STKPTR endstak(argp) STRING argp; {  
    /* For use after 'locstak' to hand back new stack top and then  
     * allocate item  
     */  
    REG STKPTR    oldstak;  
    *argp++=0; oldstak=stakbot;  
    stakbot=staktop=(STKPTR)round(argp,BYTESPERWORD);  
    return(oldstak);  
}
```

```
VOID    tdystak(x) REG STKPTR x; {  
    /* try to bring stack back to x */  
    while( ADR(stakbsy)>ADR(x) ){  
        free(stakbsy);  
        stakbsy = stakbsy->word;  
    }  
    staktop=stakbot=max(ADR(x),ADR(stakbas));  
    rmttemp(x);  
}
```

```
VOID stakchk() {  
    if( (brkend-stakbas)>BRKINCR+BRKINCR ){  
        setbrk(-BRKINCR);  
    }  
}
```